ARL-TN-0782 ● Sep 2016

US Army Research Laboratory

Android: Call C Functions with the Native Development Kit (NDK)

by Hao Q Vu

ARL-TN-0782 ● SEP 2016

US Army Research Laboratory

Android: Call C Functions with the Native Development Kit (NDK)

by Hao Q Vu
Sensors and Electron Devices Directorate, ARL

REPORT DOCUMENTATION PAGE			*Form Approved* *OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing the burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. **PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.**			
1. REPORT DATE *(DD-MM-YYYY)* September 2016	**2. REPORT TYPE** Technical Note		**3. DATES COVERED (From - To)** 02/2016–05/2016
4. TITLE AND SUBTITLE Android: Call C Functions with the Native Development Kit (NDK)			**5a. CONTRACT NUMBER**
			5b. GRANT NUMBER
			5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Hao Q Vu			**5d. PROJECT NUMBER**
			5e. TASK NUMBER
			5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) US Army Research Laboratory ATTN: RDRL-SES-P 2800 Powder Mill Road Adelphi, MD 20783-1138			**8. PERFORMING ORGANIZATION REPORT NUMBER** ARL-TN-0782
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)			**10. SPONSOR/MONITOR'S ACRONYM(S)**
			11. SPONSOR/MONITOR'S REPORT NUMBER(S)
12. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.			
13. SUPPLEMENTARY NOTES			
14. ABSTRACT The Android Native Development Kit (NDK) provides a Java Android application a simplified mechanism to call embedded C native codes, which are used to increase the performance of a computationally intensive application. This step-by-step guide is intended to assist programmers with how to attach an NDK plugin to an Android Integrated Development Environment and how to call C functions from a Java application.			
15. SUBJECT TERMS Android, NDK, Native Development Kit, C callable, Java Native Interface, JNI, Java, C/C++			

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Hao Q Vu
a. REPORT Unclassified	**b. ABSTRACT** Unclassified	**c. THIS PAGE** Unclassified	UU	20	**19b. TELEPHONE NUMBER (Include area code)** 301-394-5324

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures **iv**

1. Introduction **1**

2. Android Studio IDE **1**

3. Android NDK Package **1**

3.1 Method 1 1

3.2 Method 2 2

4. Configure NDK with Android Studio IDE 1.5.1 **2**

5. Create an Android Application **6**

6. Final Build and Run Android Application **13**

7. Conclusion **13**

Distribution List **14**

List of Figures

Fig. 1 Create Tool window ... 3
Fig. 2 “Javah” path ... 3
Fig. 3 Environment path ... 5
Fig. 4 NDK-BUILD CLEAN configuration ... 6
Fig. 5 Android application ... 6
Fig. 6 Java codes to add 2 integer numbers ... 7
Fig. 7 Native library Java class ... 7
Fig. 8 JNI directory ... 8
Fig. 9 JNI header file ... 9
Fig. 10 Create a C source file ... 10
Fig. 11 Java native C functions ... 10
Fig. 12 Android.mk makefile ... 11
Fig. 13 Application.mk makefile ... 12

1. Introduction

With its compact form factor, portability, low power consumption, and increasing processing power, the Android mobile platform is gaining popularity as a replacement for traditional desktop or laptop computers as a processing platform. Most digital signal processing algorithms developed at the US Army Research Laboratory are written in the C programming language to provide sufficient processing power to satisfy typical processing and performance requirements. Typical applications developed for the Android mobile platform are written in Java. Therefore, to achieve maximum speed, there is a need to bridge Java-based applications to native C applications. Fortunately, a combination of the Android Native Development Kit (NDK) and the Java Native Interface (JNI) provides such a mechanism.

NDK is a toolset that allows users to build C/C++ functions into a static or dynamic library or let the existing prebuilt library be called from an Android application. JNI defines a way for managed code written in Java to interact with native C/C++.

This guide is intended to take programmers through adding an NDK package into an Android Studio Integrated Development Environment (IDE), to building a simple Android application that calls a C function to add 2 integer values received from the user screen.

2. Android Studio IDE

This guide assumes the user has already downloaded the Android Studio IDE and has an extensive working knowledge of this development package. As of the writing of this report, the version of Android Studio IDE was 1.5.1.

3. Android NDK Package

The Android NDK package can be downloaded via either of the following methods.

3.1 Method 1

The latest version of the NDK package for one's operating system can be downloaded directly from "developer.android.com/ndk/downloads/index.html".

Expand the package once it has downloaded.

3.2 Method 2

Download the NDK package directly from Android Studio IDE:

1) Select the “Tools” tab.
2) Select “Android->SDK manager”.
3) Select “Appearance->System Settings->Android SDK”.
4) Select the “SDK Tools” tab.
5) Check the box labeled “Android NDK”.
6) Select “Apply”.

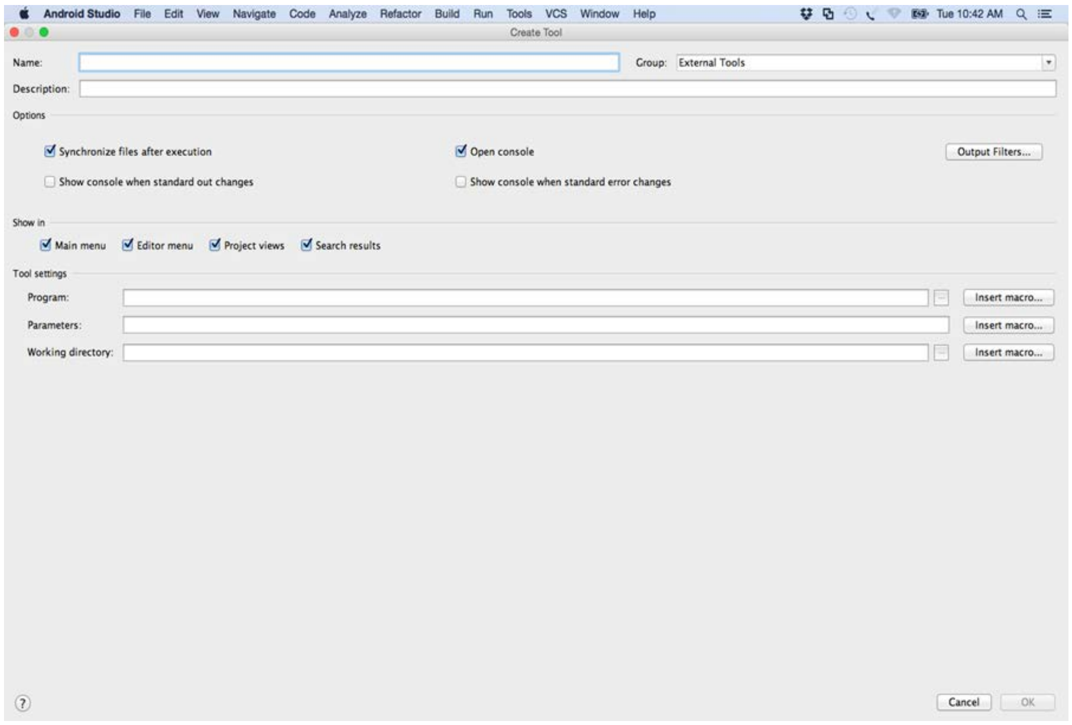
Wait for the package to be added into the Android Studio IDE. The downloaded package can be found at “~/Library/Android/sdk/ndk-bundle”.

4. Configure NDK with Android Studio IDE 1.5.1

Perform the following steps to configure the NDK with Android Studio IDE, version 1.5.1:

1) Set up the Android NDK location:
   a) Select “File->Project Structure”.
   b) Select “SDK Location”.
   c) If Method 1 was chosen, enter the NDK location of where the downloaded NDK package was expanded. If Method 2 was chosen, then Android Studio will automatically fill in the NDK location.
2) Add the JAVAH, NDK-BUILD, and NDK-BUILD CLEAN paths:

   JAVAH is a tool provided by Java SE to generate a C header and the source files that are needed to implement native methods. Refer to http://docs.oracle.com/javase/7/docs/technotes/tools/windows/javah.html for a detailed description.
3) Configure the JAVAH path:
   a) Select “Tools->Android->SDK Manager”.
   b) Select “Tools->External Tools”.
   c) Select “+” at the bottom of the right pane.
   d) The “Create Tool” window (Fig. 1) will pop up.



Name:

Group: External Tools

Description:

Options

- [✓] Synchronize files after execution
- [✓] Open console
- [ ] Show console when standard out changes
- [ ] Show console when standard error changes

Show in

- [✓] Main menu
- [✓] Editor menu
- [✓] Project views
- [✓] Search results

Tool settings

Program:

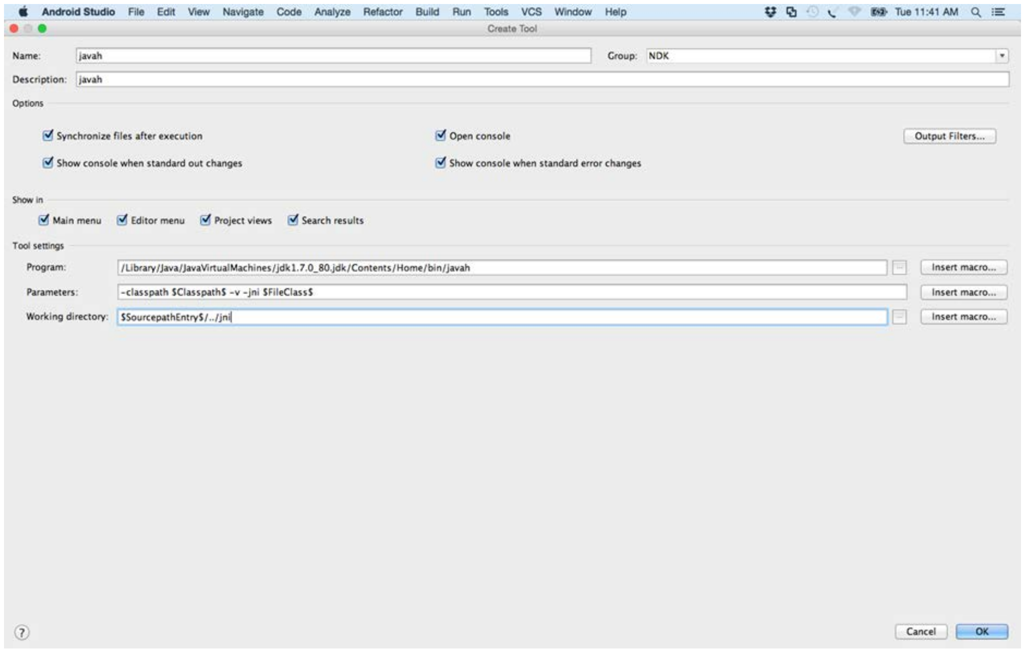
Parameters:

Working directory:

Android Studio File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
Tue 10:42 AM
Create Tool
Output Filters...
Insert macro...
Cancel
OK

Fig. 1 Create Tool window

e) Enter the following to create a path for "javah", as shown in Fig. 2.



Name: javah

Group: NDK

Description: javah

Options

- [✓] Synchronize files after execution
- [✓] Open console
- [✓] Show console when standard out changes
- [✓] Show console when standard error changes

Show in

- [✓] Main menu
- [✓] Editor menu
- [✓] Project views
- [✓] Search results

Tool settings

Program: /Library/Java/JavaVirtualMachines/jdk1.7.0_80.jdk/Contents/Home/bin/javah

Parameters: -classpath $Classpath$ -v -jni $FileClass$

Working directory: $SourcepathEntry$/../jni

Android Studio File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
Tue 11:41 AM
Create Tool
Output Filters...
Insert macro...
Cancel
OK

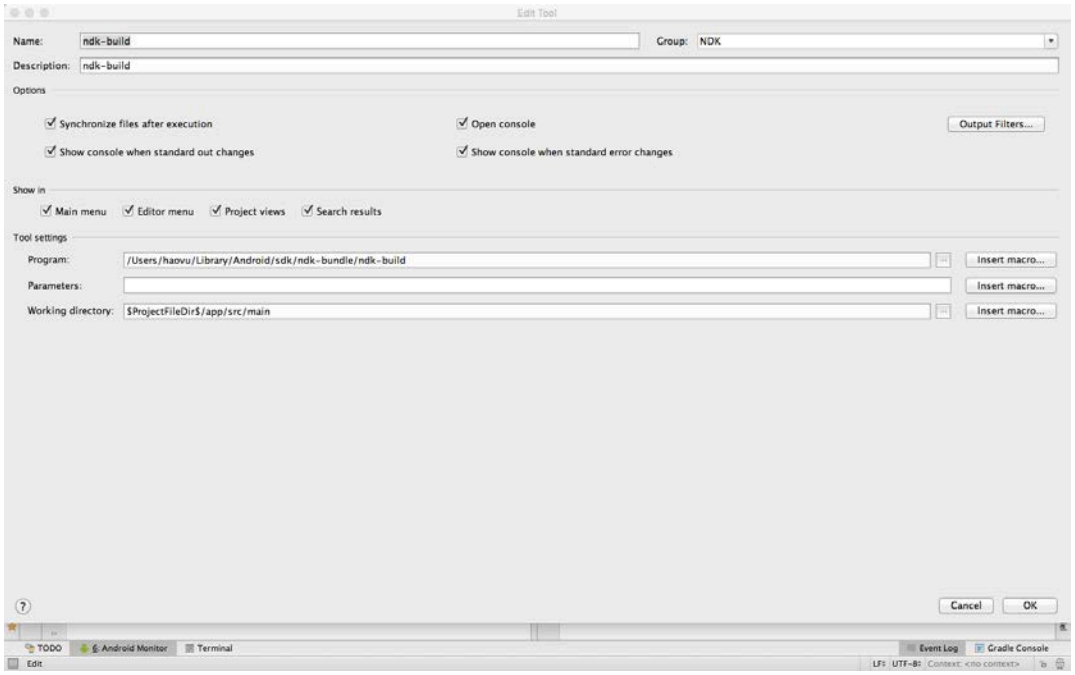
Fig. 2 "Javah" path

Notes:

- *Program*: This needs to point to where the JAVAH application is located on the host.
- *Parameters*: The parameters are set to -classpath $Classpath$ -v –jni $FileClass$
    - -classpath specifies the path for "javah" to look for classes
    - -v verbose
    - -jni tells "javah" to create an output file with JNI-style native method function prototypes
- *Working directory*: All the required files needed to create a C callable object are located in the "jni" directory.

NDK-BUILD is a shell script used to call a GNU "make" 3.81 or later. Use the following steps to configure it:

1) Configure the NDK-BUILD path:
    a) Select "Tools->Android->SDK Manager".
    b) Select "Tools->External Tools.
    c) Select "+" at the bottom of the right pane.
    d) The "Create Tool" window (see Fig. 1) will pop up.
    e) Enter the following to create an environment path for NDK-BUILD, as shown in Fig. 3:
        - *Program*: This points to the location to where NDK-BUILD is located.
        - *Parameters*: Leave blank.
        - *Working directory*: This points to where all the Java files are located.



Name: ndk-build
Group: NDK
Description: ndk-build

Options
- ✓ Synchronize files after execution
- ✓ Open console
- ✓ Show console when standard out changes
- ✓ Show console when standard error changes

Show in
- ✓ Main menu
- ✓ Editor menu
- ✓ Project views
- ✓ Search results

Tool settings
- Program: /Users/haovu/Library/Android/sdk/ndk-bundle/ndk-build
- Parameters:
- Working directory: $ProjectFileDir$/app/src/main

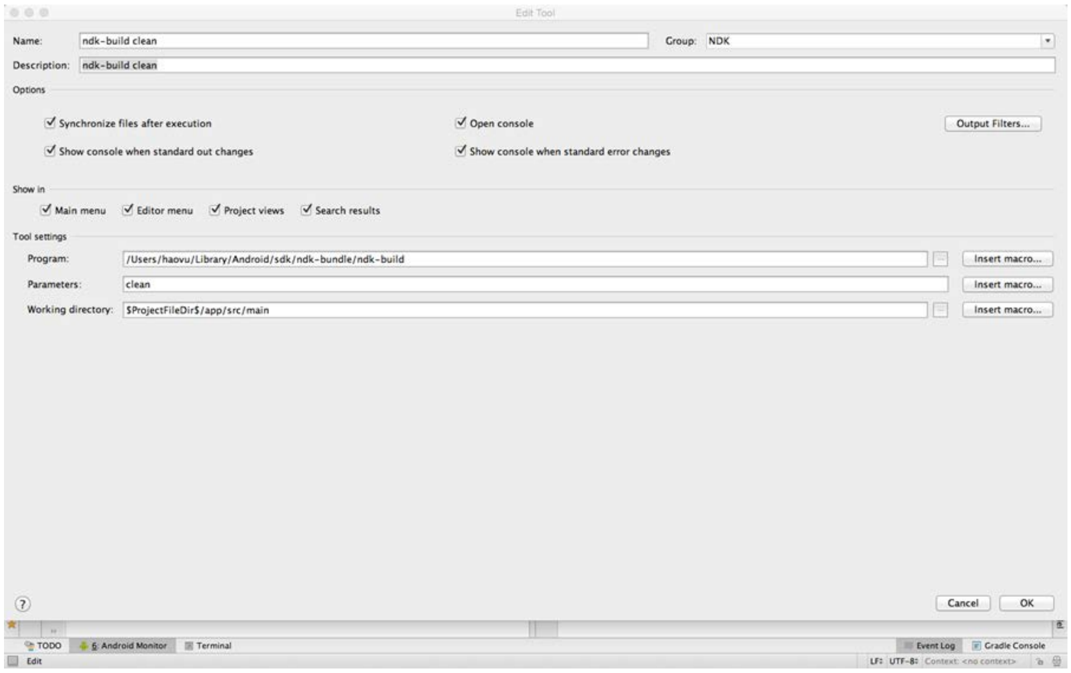
Fig. 3 Environment path

NDK-BUILD CLEAN is used to remove all previously generated binaries. Use the following steps to configure it:

1) Configure the NDK-BUILD CLEAN path:

    a) Follow the same steps as if configuring the NDK-BUILD path, except the parameter is set to “clean”.

    b) The final NDK-BUILD CLEAN configuration screen should look like Fig. 4.



Name: ndk-build clean
Group: NDK
Description: ndk-build clean

Options
- ✓ Synchronize files after execution
- ✓ Open console
- ✓ Show console when standard out changes
- ✓ Show console when standard error changes

Show in
- ✓ Main menu
- ✓ Editor menu
- ✓ Project views
- ✓ Search results

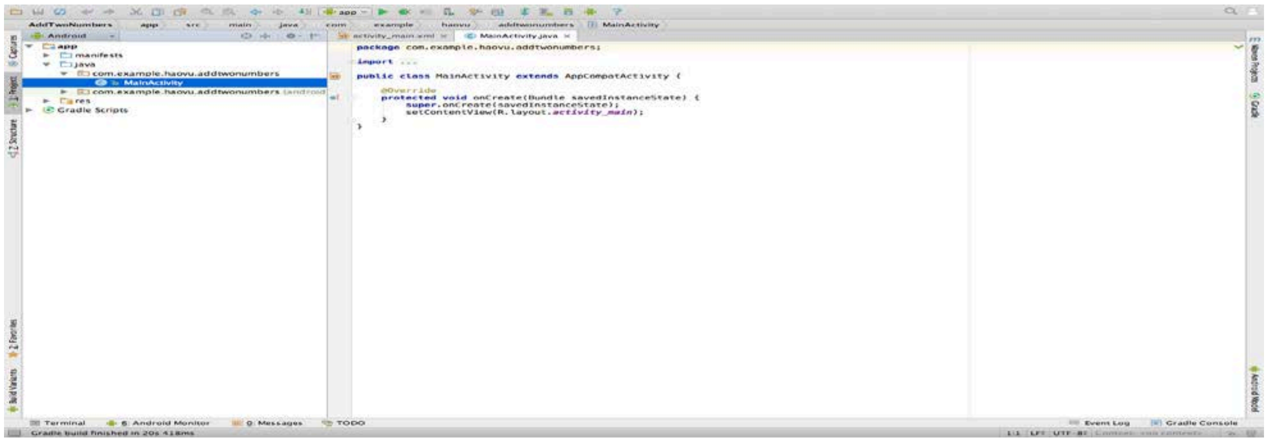
Tool settings
Program: /Users/haovu/Library/Android/sdk/ndk-bundle/ndk-build
Parameters: clean
Working directory: $ProjectFileDir$/app/src/main

Fig. 4 NDK-BUILD CLEAN configuration

5. Create an Android Application

This guide assumes that the user is well versed in using the Android Studio IDE to create an Android application. Therefore, a step-by-step guide on how to create an Android application has not been included. A simple Android application is provided as an example highlighting the functionality outlined in this report.

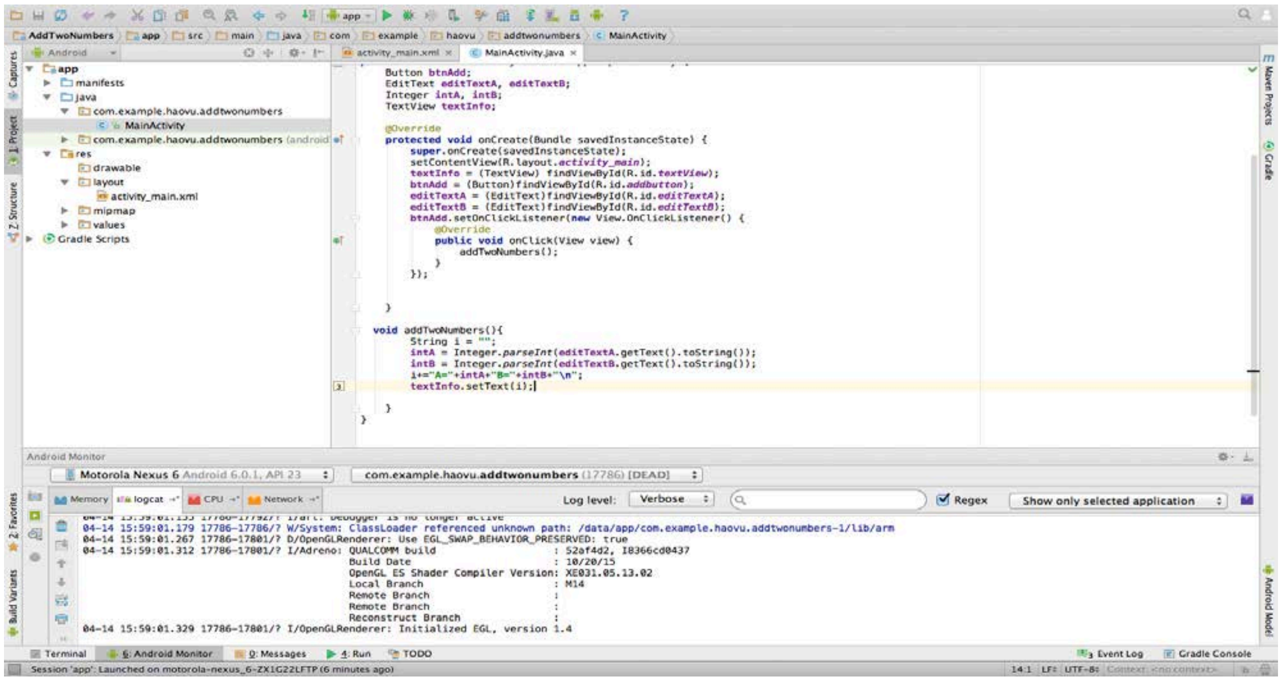
A newly created Android application is shown is Fig. 5.



```
package com.example.haovu.addtwonumbers;

import ...

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

Fig. 5 Android application

The user can then add the ability to read in 2 integer numbers, as shown in Fig. 6.



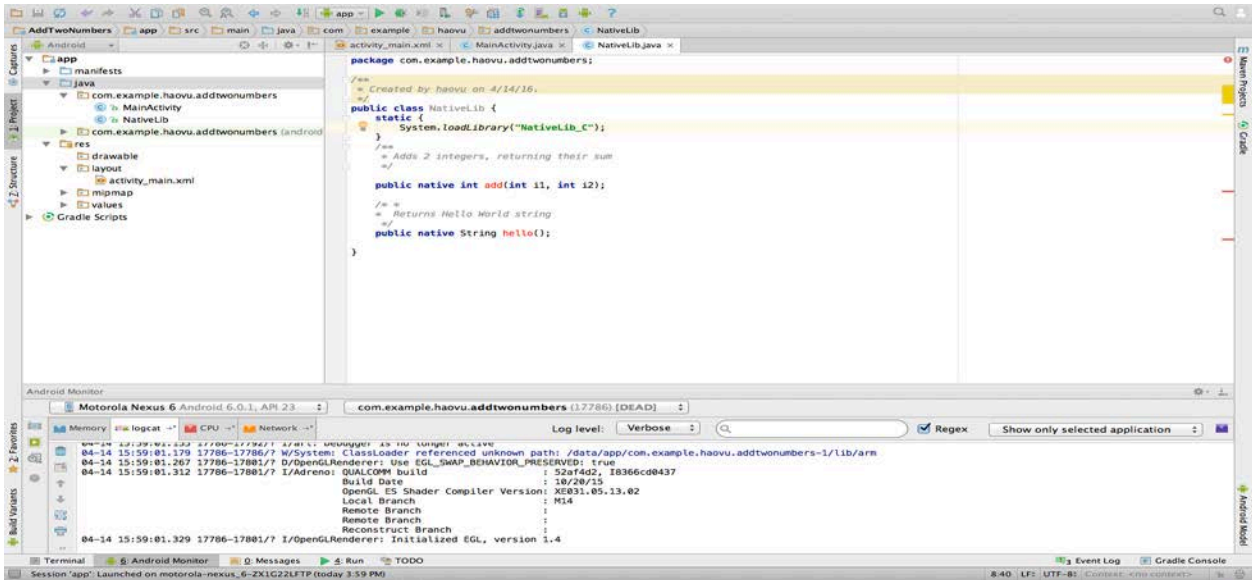
```
    Button btnAdd;
    EditText editTextA, editTextB;
    Integer intA, intB;
    TextView textInfo;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        textInfo = (TextView) findViewById(R.id.textView);
        btnAdd = (Button)findViewById(R.id.addbutton);
        editTextA = (EditText)findViewById(R.id.editTextA);
        editTextB = (EditText)findViewById(R.id.editTextB);
        btnAdd.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                addTwoNumbers();
            }
        });


    }

  void addTwoNumbers(){
        String i = "";
        intA = Integer.parseInt(editTextA.getText().toString());
        intB = Integer.parseInt(editTextB.getText().toString());
        i+="A="+intA+"B="+intB+"\n";
        textInfo.setText(i);

    }
}
```

```
04-14 15:59:01.179 17786-17786/? W/System: ClassLoader referenced unknown path: /data/app/com.example.haovu.addtwonumbers-1/lib/arm
04-14 15:59:01.267 17786-17801/? D/OpenGLRenderer: Use EGL_SWAP_BEHAVIOR_PRESERVED: true
04-14 15:59:01.312 17786-17801/? I/Adreno: QUALCOMM build                   : 52af4d2, I8366cd0437
                                           Build Date                       : 10/20/15
                                           OpenGL ES Shader Compiler Version: XE031.05.13.02
                                           Local Branch                     : M14
                                           Remote Branch                    :
                                           Remote Branch                    :
                                           Reconstruct Branch               :
04-14 15:59:01.329 17786-17801/? I/OpenGLRenderer: Initialized EGL, version 1.4
```

Fig. 6 Java codes to add 2 integer numbers

To do this, the user must set up a sequence to call a C function from an Android application:

1) Create a native library Java class as shown in Fig. 7. This class resides in the "src "directory. This class includes an Android system call to load the library that was written in C and all the C function prototypes.



```
package com.example.haovu.addtwonumbers;

/**
 * Created by haovu on 4/14/16.
 */
public class NativeLib {
    static {
        System.loadLibrary("NativeLib_C");
    }
    /**
     * Adds 2 integers, returning their sum
     */

    public native int add(int i1, int i2);

    /* *
    *  Returns Hello World string
     */
    public native String hello();

}
```

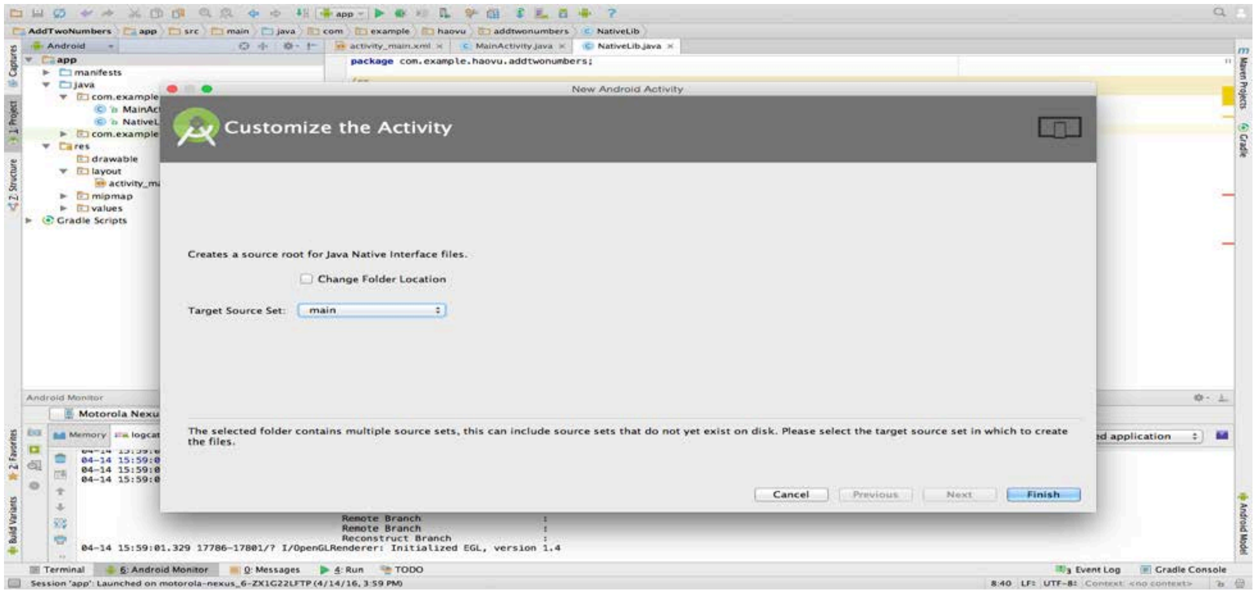
```
04-14 15:59:01.179 17786-17786/? W/System: ClassLoader referenced unknown path: /data/app/com.example.haovu.addtwonumbers-1/lib/arm
04-14 15:59:01.267 17786-17801/? D/OpenGLRenderer: Use EGL_SWAP_BEHAVIOR_PRESERVED: true
04-14 15:59:01.312 17786-17801/? I/Adreno: QUALCOMM build                   : 52af4d2, I8366cd0437
                                           Build Date                       : 10/20/15
                                           OpenGL ES Shader Compiler Version: XE031.05.13.02
                                           Local Branch                     : M14
                                           Remote Branch                    :
                                           Remote Branch                    :
                                           Reconstruct Branch               :
04-14 15:59:01.329 17786-17801/? I/OpenGLRenderer: Initialized EGL, version 1.4
```

Fig. 7 Native library Java class

2) Create the JNI directory as shown in Fig. 8, to store all related C source files and H header files:

   a) Select “app->New->Folder->JNI folder”.

   b) Select “Finish”. The “jni” directory should appear under the “app” directory.



```
package com.example.haovu.addtwonumbers;
```

Creates a source root for Java Native Interface files.

Change Folder Location

Target Source Set: main

The selected folder contains multiple source sets, this can include source sets that do not yet exist on disk. Please select the target source set in which to create the files.

```
04-14 15:59:01.329 17786-17801/? I/OpenGLRenderer: Initialized EGL, version 1.4
```

Session 'app': Launched on motorola-nexus_6-ZX1G22LFTP (4/14/16, 3:59 PM)

Fig. 8 JNI directory

3) Create the JNI header file using the external tool JAVAH, as shown in Fig. 9:

   a) Select “NativeLib->NDK->javah”:

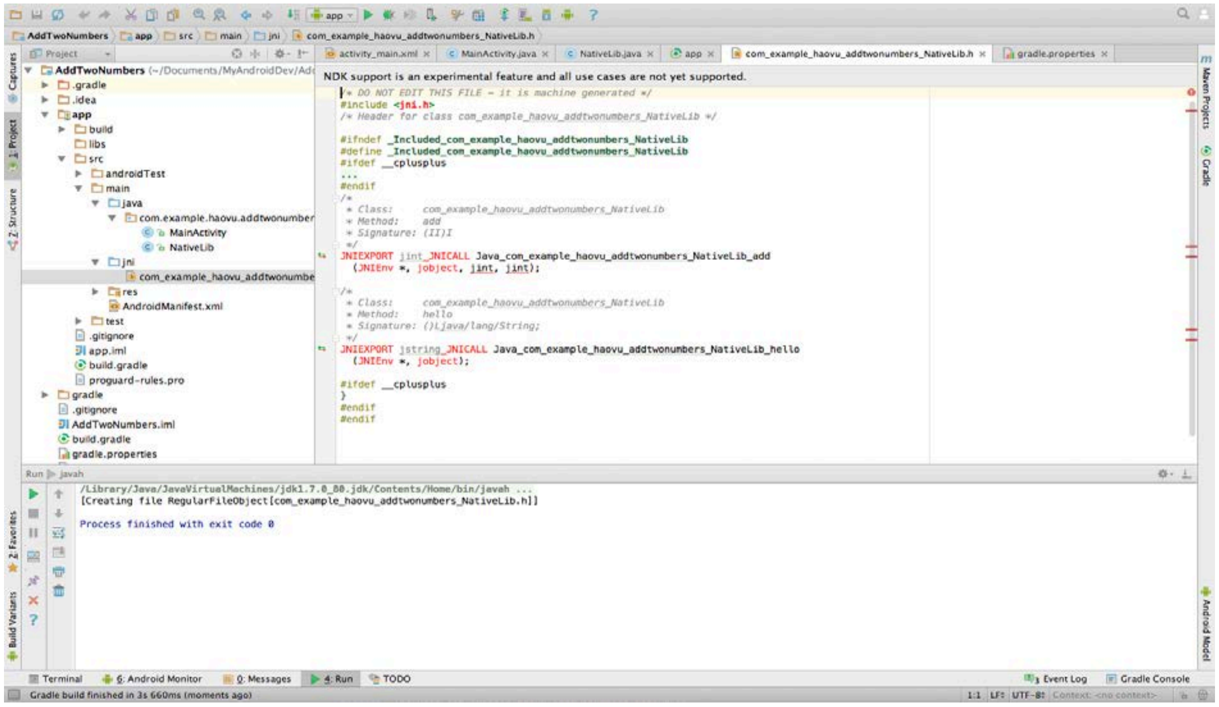
      Based on the Java class created earlier, a header file will appear under the “jni” directory that has the following naming convention.

      - PackageName: *com_example_haovu_addtwonumbers*
      - Followed by the NativeJavaClassname: *NativeLib*
      - Followed by the *.h* extension
      - Example: *com_example_haovu_ addtwonumbers_NativeLib.h*

   b) The newly created header file contains the Java native function prototypes with the following format:

      - *JNIEXPORT*
      - The function return type

- *JNICALL*
- *Java*
- Followed by the PackageName
- Followed by the NativeJavaClassname
- Followed by the FunctionName
- Example: *JNIEXPORT jint JNICALL Java_com_example_haovu_addtwonumbers_NavtiveLb add (JNIEnv *, jobject, jint, jint)*



NDK support is an experimental feature and all use cases are not yet supported.

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class com_example_haovu_addtwonumbers_NativeLib */

#ifndef _Included_com_example_haovu_addtwonumbers_NativeLib
#define _Included_com_example_haovu_addtwonumbers_NativeLib
#ifdef __cplusplus
...
#endif
/*
 * Class:     com_example_haovu_addtwonumbers_NativeLib
 * Method:    add
 * Signature: (II)I
 */
JNIEXPORT jint JNICALL Java_com_example_haovu_addtwonumbers_NativeLib_add
  (JNIEnv *, jobject, jint, jint);

/*
 * Class:     com_example_haovu_addtwonumbers_NativeLib
 * Method:    hello
 * Signature: ()Ljava/lang/String;
 */
JNIEXPORT jstring JNICALL Java_com_example_haovu_addtwonumbers_NativeLib_hello
  (JNIEnv *, jobject);

#ifdef __cplusplus
}
#endif
#endif
```

```
/Library/Java/JavaVirtualMachines/jdk1.7.0_80.jdk/Contents/Home/bin/javah ...
[Creating file RegularFileObject[com_example_haovu_addtwonumbers_NativeLib.h]]

Process finished with exit code 0
```

Gradle build finished in 3s 660ms (moments ago)

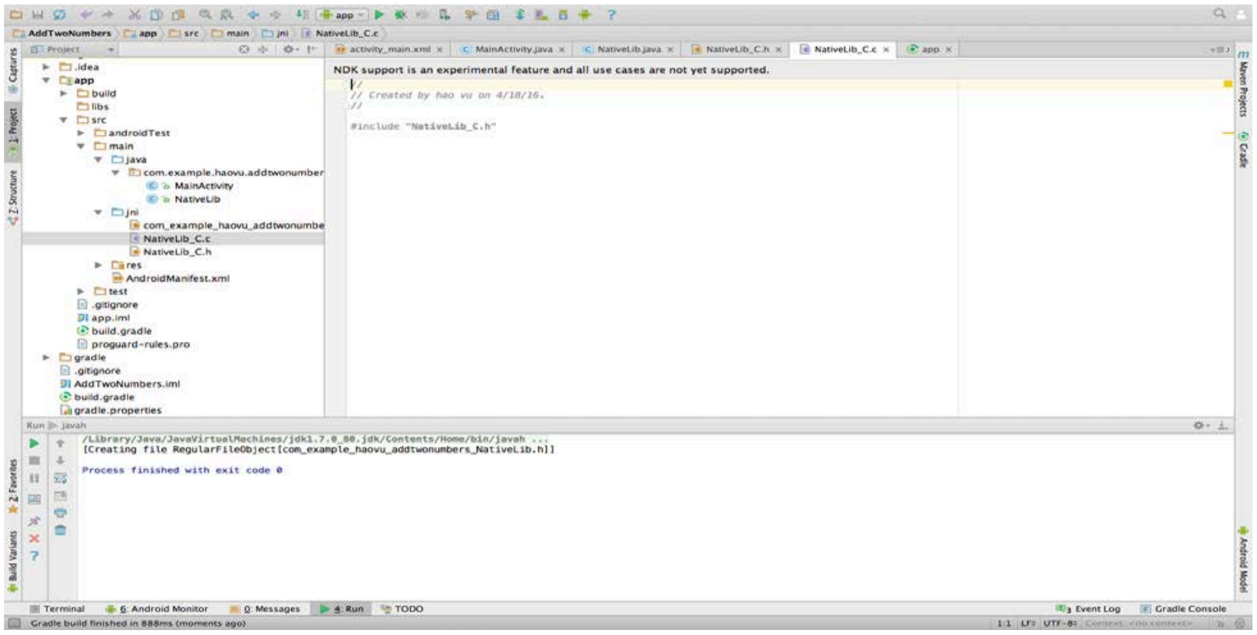
Fig. 9 JNI header file

c) Under the "jni" directory, create a C source file (NativeLib_C.c), as shown in Fig. 10, to implement the native C functions:

   i) Select "jni->New->C/C++ Source File".

   ii) Enter the filename and type.

   iii) Select "OK".



NDK support is an experimental feature and all use cases are not yet supported.

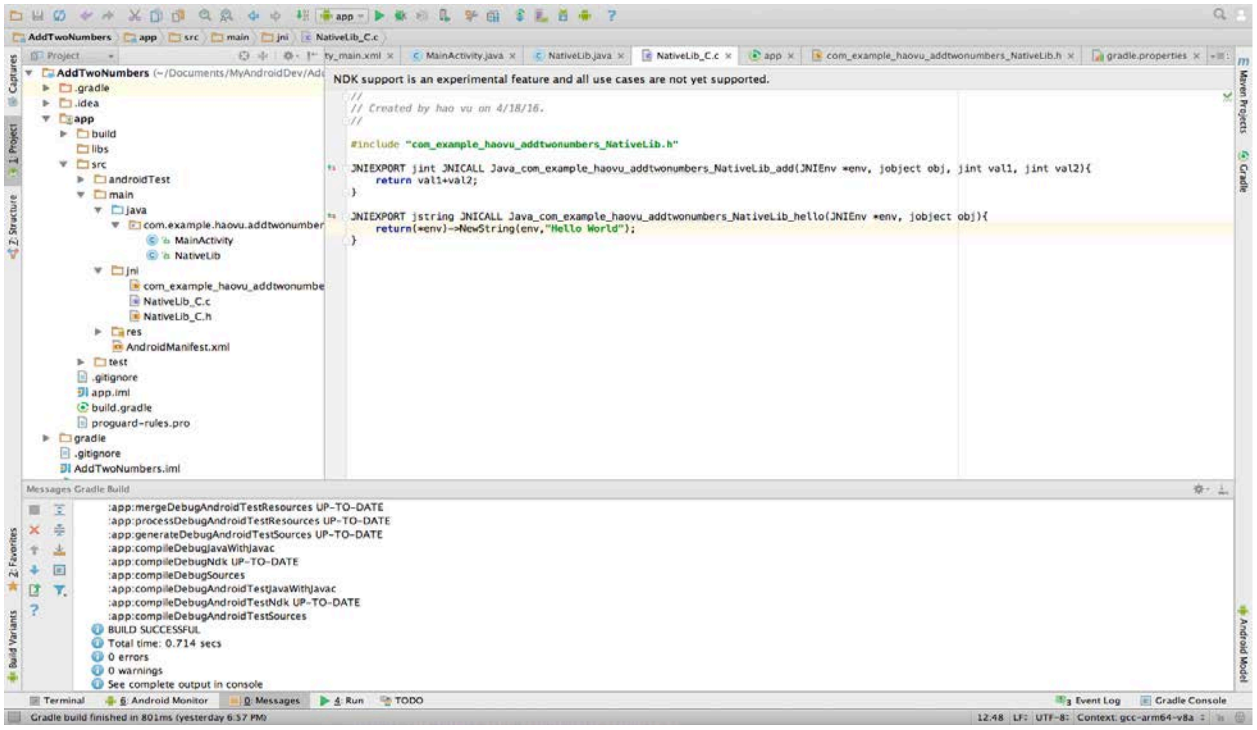
```
//
// Created by hao vu on 4/18/16.
//

#include "NativeLib_C.h"
```

```
/Library/Java/JavaVirtualMachines/jdk1.7.0_80.jdk/Contents/Home/bin/javah ...
[Creating file RegularFileObject[com_example_haovu_addtwonumbers_NativeLib.h]]

Process finished with exit code 0
```

Gradle build finished in 888ms (moments ago)

Fig. 10 Create a C source file

d) Implement the Java native C functions as shown in Fig. 11.



NDK support is an experimental feature and all use cases are not yet supported.

```
//
// Created by hao vu on 4/18/16.
//

#include "com_example_haovu_addtwonumbers_NativeLib.h"

JNIEXPORT jint JNICALL Java_com_example_haovu_addtwonumbers_NativeLib_add(JNIEnv *env, jobject obj, jint val1, jint val2){
    return val1+val2;
}

JNIEXPORT jstring JNICALL Java_com_example_haovu_addtwonumbers_NativeLib_hello(JNIEnv *env, jobject obj){
    return(*env)->NewString(env,"Hello World");
}
```

```
:app:mergeDebugAndroidTestResources UP-TO-DATE
:app:processDebugAndroidTestResources UP-TO-DATE
:app:generateDebugAndroidTestSources UP-TO-DATE
:app:compileDebugJavaWithJavac
:app:compileDebugNdk UP-TO-DATE
:app:compileDebugSources
:app:compileDebugAndroidTestJavaWithJavac
:app:compileDebugAndroidTestNdk UP-TO-DATE
:app:compileDebugAndroidTestSources
BUILD SUCCESSFUL
Total time: 0.714 secs
0 errors
0 warnings
See complete output in console
```

Gradle build finished in 801ms (yesterday 6:57 PM)

Fig. 11 Java native C functions

e) Modify “build.gradle (Module:app)”:

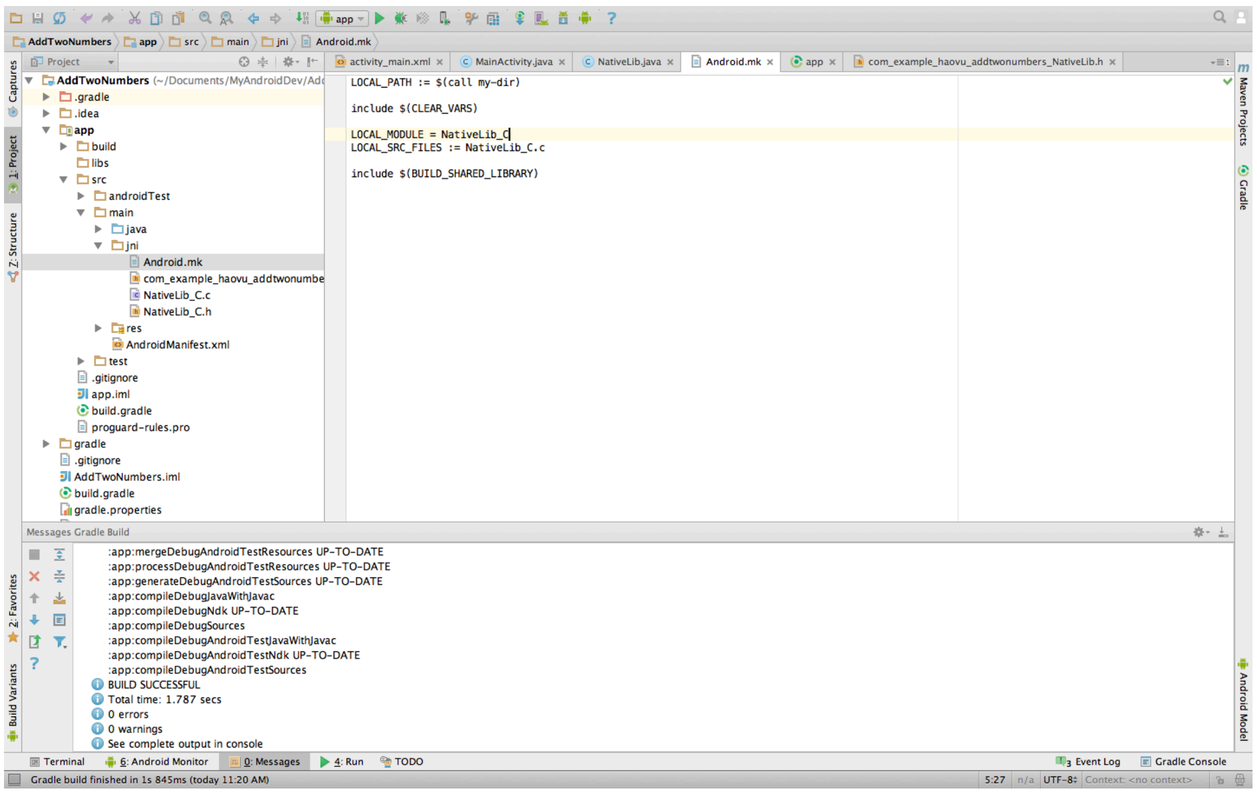
   i) Inside the “defaultConfig” section, add the following pseudo code:

```
sourceSets {
    main{
        jni.srcDirs = []
        jniLibs.srcDir “src/main/libs”
    }
}
```

f) Create an “Android.mk” makefile, shown in Fig. 12, under the “jni” directory to determine how to build the C code:

   i) Select “jni->New->File”.

   ii) Enter “Android.mk” as the filename.



```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

LOCAL_MODULE = NativeLib_C
LOCAL_SRC_FILES := NativeLib_C.c

include $(BUILD_SHARED_LIBRARY)
```

```
:app:mergeDebugAndroidTestResources UP-TO-DATE
:app:processDebugAndroidTestResources UP-TO-DATE
:app:generateDebugAndroidTestSources UP-TO-DATE
:app:compileDebugJavaWithJavac
:app:compileDebugNdk UP-TO-DATE
:app:compileDebugSources
:app:compileDebugAndroidTestJavaWithJavac
:app:compileDebugAndroidTestNdk UP-TO-DATE
:app:compileDebugAndroidTestSources
BUILD SUCCESSFUL
Total time: 1.787 secs
0 errors
0 warnings
See complete output in console
```

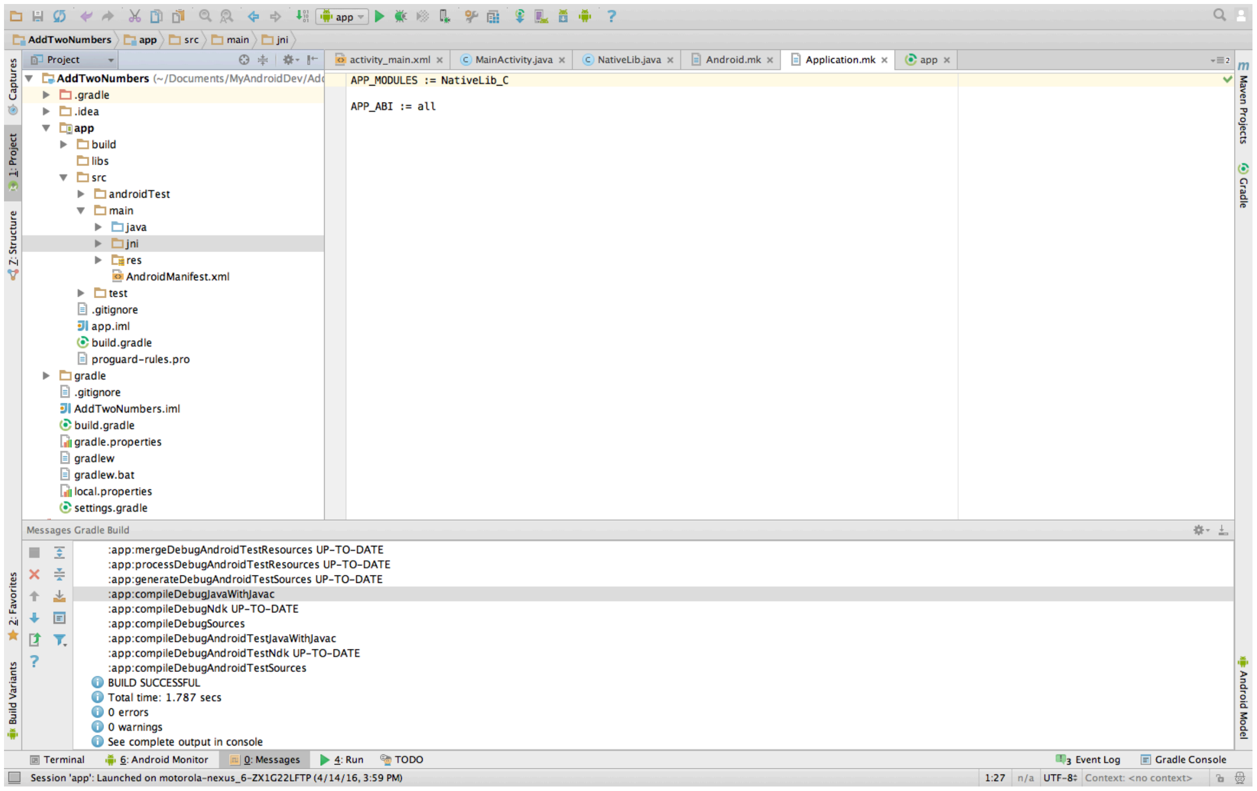
Gradle build finished in 1s 845ms (today 11:20 AM)

Fig. 12 Android.mk makefile

g) Create the "Application.mk" makefile, shown in Fig. 13, under the "jni" directory to tell the NDK what architecture it should build the shared library for:

   i) Select "jni->New->File".

   ii) Enter "Application.mk" as the filename.



```
APP_MODULES := NativeLib_C

APP_ABI := all
```

```
:app:mergeDebugAndroidTestResources UP-TO-DATE
:app:processDebugAndroidTestResources UP-TO-DATE
:app:generateDebugAndroidTestSources UP-TO-DATE
:app:compileDebugJavaWithJavac
:app:compileDebugNdk UP-TO-DATE
:app:compileDebugSources
:app:compileDebugAndroidTestJavaWithJavac
:app:compileDebugAndroidTestNdk UP-TO-DATE
:app:compileDebugAndroidTestSources
BUILD SUCCESSFUL
Total time: 1.787 secs
0 errors
0 warnings
See complete output in console
```

```
Session 'app': Launched on motorola-nexus_6-ZX1G22LFTP (4/14/16, 3:59 PM)
```

Fig. 13 Application.mk makefile

Note: This application.mk requests the NDK to build a shared library for all supported architectures. The following details how to build a shared library:

1) Select "app->NDK->ndk-build":

   - A "libs" directory that stores the shared library for all the architectures is automatically created.
   - All the supported architecture directories are created.
   - Under each architecture directory, a shared library for that particular architecture can be found.

6. Final Build and Run Android Application

Perform the following steps to create the final build and run the Android application:

1) Select “Build->Make Project”.
2) Select “Run-> Run ‘app’”.

7. Conclusion

This guide summarizes all the necessary mechanisms and steps to guide a novice Android application developer to build an Android application that is capable of handling a high-intensive computation requirement.

1	DEFENSE TECHNICAL
(PDF)	INFORMATION CTR
	DTIC OCA
2	DIRECTOR
(PDF)	US ARMY RESEARCH LAB
	RDRL CIO L
	IMAL HRA MAIL & RECORDS
	MGMT
1	GOVT PRINTG OFC
(PDF)	A MALHOTRA
8	US ARMY RSRCH LAB
(PDF)	RDRL SES P
	C REIFF
	D GONSKI
	H VU
	L SIM
	M SCANLON
	S TENNEY
	W ALBERTS II
	RDRL SES S
	B MAYS